Day : Wednesday

PALM INTRANET

Date: 5/12/2004
Time: 12:08:07

Inventor Name Search Result

Your Search was:

Last Name = KERNE
First Name = ANDRUID

Application#	Patent#	Status	Date Filed	Title	Inventor Name 2
60182319	Not Issued	159	02/14/2000	WEIGHTED INTERACTIVE GRID PRESENTATION SYSTEM AND METHOD FOR STREAMING MULTIMEDIA COLLAGE	KERNE, ANDRUID
09784369	Not Issued	030	02/14/2001	WEIGHTED INTERACTIVE GRID PRESENTATION SYSTEM AND METHOD FOR STREAMING A MULTIMEDIA COLLAGE	KERNE, ANDRUID

Inventor Search Completed: No Records to Display.

Search Another: Inventor

Last Name	First Name	
KERNE	ANDRUID	Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

CHI 96
Electronic Proceedings

Home
Indexes
Copyright

Papers
Contents
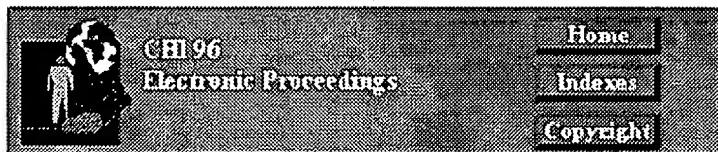
The WebBook and the Web Forager: An Information Workspace for the World-Wide Web

Stuart K. Card, George G. Robertson, and William York

Xerox Palo Alto Research Center 3333 Coyote Hill Road Palo Alto, California 94304
E-mail: {card | robertson | york}@parc.xerox.com

ABSTRACT

The World-Wide Web has achieved global connectivity stimulating the transition of computers from knowledge processors to knowledge sources. But the Web and its client software are seriously deficient for supporting users' interactive use of this information. This paper presents two related designs with which to evolve the Web and its clients. The first is the WebBook, a 3D interactive book of HTML pages. The WebBook allows rapid interaction with objects at a higher level of aggregation than pages. The second is the Web Forager, an application that embeds the WebBook and other objects in a hierarchical 3D workspace. Both designs are intended as exercises to play off against analytical studies of information workspaces.

Keywords

3D graphics, user interfaces, information access, World-Wide Web, information workspace, workspace.

INTRODUCTION

Whereas personal computers used to be viewed as knowledge processorsÑword processors or spreadsheet calculators, for instance, they are now becoming viewed as knowledge sourcesÑportals to vast stores of information on-line or on CD-ROMs [1]. This is true because much work has become knowledge work and because the infrastructure for distributed knowledge access has been developing. The most dramatic development of this infrastructure has been the growth of the World Wide Web in the last couple of years. Uniform access protocols have enabled a person with a personal computer and a communications link access by button-click to millions of pages around the world.

Despite the exhilaration felt by many users at this achievement of wide-scale connectivity, there are problems that call for evolution of the medium: Pages are often hard to find, users get lost and have difficulty relocating previously-found pages, they have difficulty organizing things once found,

difficulty doing knowledge processing on the found thing, and interacting with the Web is notoriously too slow to incorporate it gracefully into human activity. In this paper, we suggest a way of viewing the Web and its problems, then propose two related innovations, the WebBook[TM] and the Web Forager [TM], to mitigate these problems.

INFORMATION FORAGING ON THE WEB

In an information-rich world, the limiting quantity for users isnÕt so much the information in the world as the userÕs own limited time. Just as animals forage for food and try to optimize their food rate of gain, users often seek strategies to optimize their information gain per unit time and in fact, we can make the analogy literal by thinking of the Web in terms of Information Foraging Theory [1], an analogue of foraging models from ecological biology [2].

In terms of this theory, the user stalks certain types of information. In a particular environment, this sort of information is encountered at a certain rate of l relevant pages/hour, say. The Web is an evolving information ecology in which on the one hand users are trying to evolve methods to increase the encounter rates of relevant information and on the other hand information sources are trying to evolve their attractiveness to users. These result in a clumpy structure of patches of high l. Three mechanisms in particular have evolved on the server side: First, indexes, such as Lycos [3] attempt to visit and form an inverted index of every page by following all the links. The user can formulate a keyword query and obtain a patch of possible links to forage. Creation of such a patch is a form of information enrichment.

A second sort of information enrichment is a table of contents lists such as Yahoo [4]These systems provide typically a tree of categories with links of Web pages at their leaves. Again, this technique provides enriched patches for foraging. A third sort of enrichment are the home pages provided by many users, which collect together lists of related links. Again, these often provide patches with higher encounter rates. All three responses represent evolutionary adaptation (Lycos and Yahoo are successful enough that they have now been converted to businesses) and emergent self-organizing structure on the Web. Yet these innovations do not address the basic cost-structure problem of users using Web information as part of some activity.

THE COST STRUCTURE OF INFORMATION WORKSPACES

The Web maintains a uniform cost structure. The time per interaction is fast, compared to the time to, say, go to the library, but it is slow compared to interaction rates, say the time to interact with pieces of paper on a desk. Empirically, users tend to interact repeatedly with small clusters of information, a property known as locality of reference [5, 6]. As a result, information workspaces, that is, environments that are cost-tuned for doing information-based work, tend to exhibit a certain cost-structure of information: a small amount of information is organized to be available at very low cost, larger amounts are available at moderate costs, large amounts at high cost. By so doing, they capitalize on locality of reference and the activity is speeded considerably. A routine example would be a typical (ideal) office where a small amount of information is kept available on the desk; moderate amounts of information, moderately available in nearby files; and large amounts, slowly available are kept in a library down the hall. Users constantly rearrange their environments to tune the relative costs of the information, so as to make them efficient. And if they donÕt, they suffer accordingly. An important activity they do in such environments is to use them for sensemaking [7], that is, the restructuring, recoding, and analysis of information for purposes of insight.

But the Web does not exhibit the characteristics of a good information workspace. Users do not have the ability to create adequately tuned environments nor is sensemaking supported. The major effort to allow

users to organize their workspaces has been the development of variants of the "hotlist" notion. Fig. 1 show a typical example from Netscape 1.1N. User actions are provided for adding or deleting an element to a hot list, arranging an element under a heading, changing itÕs position in the list, or searching for it. Because of the interface, these mechanisms are very slow to use and do not work well with more than a couple dozen entries. Even when the entry is found, the user must still wait for the slow access times before the page appears. Hence the space is not tunable to a reasonably cost-structured workspace. Multiple windows can be spawned for access to multiple pages, but these then slow the user down because they overlap. Finally, sensemaking is impeded. In the conventional Web browsers, users are always at a particular page. But the way a user works with information is to have multiple pages simultaneously available that can be juxtaposed, rapidly accessed, and structured, such as by grouping or other layout.
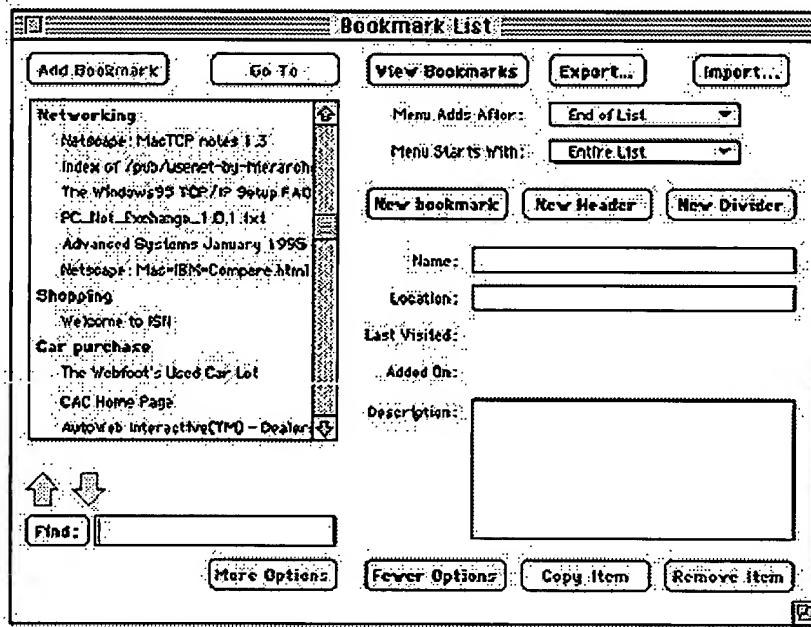
Fig. 1. Hotlist browser from Netscape.

In order to make the use of the Web better able to support information work (or for that matter, entertainment), we propose in this paper two basic moves:

- First, a move from the single Web page as the unit of interaction to a higher, aggregate entity.

We call this entity a WebBook, and it allows the user to group together related Web pages (an elementary form of sensemaking) and to manipulate these pages as a unit.

- Second, a move from a work environment containing a single element to a workspace in which the page is contained with multiple other entities, including WebBooks.

We call this environment the Web Forager. This information workspace allows for the intensive, rapid interaction among pages and allows for the assembly on the user side of hierarchical cost-structures of information necessary for the task tuning of the workspace.

Each of these has been implemented on a Silicon Graphics Iris computer using the Information Visualizer system [8]. Efforts are underway to reimplement them on a PC and to continue advancing the

design.
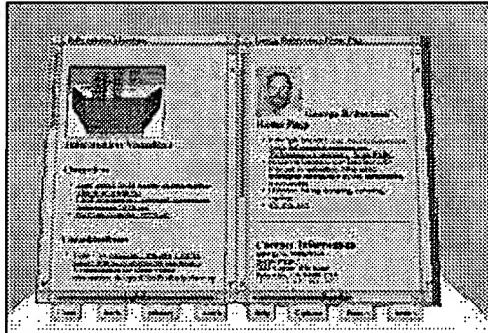
Fig. 2: Example of a WebBook

The WebBook

Current Web servers and browsers focus attention at the link and page levels. These levels are too low to represent gracefully some higher-level structures, resulting in many more entities in the Web space than are necessary and hence to orientation and sensemaking problems. We know from analysis of the web that there are structures that the user could take advantage of if the user were aware of their existence. For example, a typical home page on the web has a collection of related pages that can be reached with relative URLs (Uniform Resource Locators). It is very typical for the creator of such pages to use relative URLs instead of absolute URLs, so that the collection of pages can be physically moved easily. But current web browsers pay no attention to the distinction between relative and absolute URLs, and the user simply sees one page at a time, with no difference between a page "in the collection" and one outside the collection. Our proposal is to create a Web entity at a higher level of abstraction, a WebBook. A natural candidate structure to represent this abstraction is the book metaphor, which has been used by us [9] as well as others [10-18] previously.

Fig. 2 shows a picture of a WebBook. Given a collection of web pages, it preloads those pages and displays them as a collection using an augmented simulation of a physical book. 3D graphics and interactive animation are used to give the user a clear indication of the relationship between the pages of the book. Each page of the WebBook is a page from the web. Links are color coded so the user can easily tell the difference between a reference to another page in the book (red links) and a reference outside the book (blue links). Picking a red link will animate the flipping of pages to the desired page. Picking a blue link will close the current WebBook and look for the page elsewhere. If the page is in another WebBook stored on a bookshelf, that WebBook is opened to the desired page. If the page is in none of the WebBooks, then the Web Forager is used to display the individual page in the user's information workspace.

There are a number of features in the WebBook that make it intuitive to use. The user has several ways to flip through the pages of the book, all animated so the user can continue to see the text and images on pages while they turn. The simplest method is to click on a page (away from any link on that page); this will flip to the next or previous page depending on whether user clicked on the right or left page. The user can also click on the right or left edge of the book. The relative distance along that edge indicates how far to flip. The user can also scan the book with forward and backward scan controls (two of the buttons on the bottom of the book). The scan rate and pause time at each page is a user preference. When the user clicks on a page during a scan, the scan stops. Finally, the user can ruffle through the pages (Fig. 3) by clicking and holding the mouse button down. The ability to rapidly riffle through a set of pages has previously been a method of rapid scanning for information that could only be done with

physical books.

Fig. 3. Example of ruffling pages in a WebBook
(Click on picture to see pages move)
[418K MPEG. Should play in 2 sec.]
For QuickTime version (1.2M) click here

In addition, the user can leave a bookmark on any page. When the book is closed, a bookmark is automatically left for the last page that was viewed. Also, there is a "Back" and "History" mechanism that follows the same convention as NetScape's versions of those commands. The WebBook can be stored on a bookshelf using a simple gesture. When it is removed from the bookshelf (by clicking on it), it reopens to the last page that was being viewed.

Each page in the WebBook has three scrollbars. Two of these are the familiar vertical and horizontal scrolling controls (left and bottom scrollbars). The third (on the right) is for scaling the font size, since the trade-off of font-size vs. amount of page viewed differs for individual pages. As the scale scrollbar is moved, images remain the same size but the text changes size and is refilled continuously. A menu command allows the user to apply a new font scale to all pages of the WebBook. The corners of each page of the WebBook are resize tabs, which the user can use to change the size of the book.

Books are compact but (except for bookmarks) sequential. Therefore we allow the user to explode the book out (in animation) so that all the pages are available simultaneously. The Document Lens [19] can then be used to inspect portions of interest. Fig. 4 shows the WebBook Document Lens view. The user is then able to pan and zoom over the entire set of pages, while retaining a focus plus context display of the book. When returning to the book view, the user sees an animation of the pages imploding into a book.

WebBook Applications

The WebBook can be used to view any collection of web pages. The principle difference is the method used to generate the URLs. The method used to collect URLs leads to a number of applications.

Relative-URL Books. One interesting choice of pages is based on recursively finding all relative URLs starting from a given page. These pages are intrinsically related because their creator wanted to be able to move them as a group. We have found this heuristic often produces coherent and interesting books. *Home-Page Books.* Probably the simplest choice of pages is those pages referred to directly from a given page. Users throughout the net have strongly expressed their desire to make sense of the net by collecting sets of related URLs. The WebBook goes the next step and allows the collection of pages to be rapidly consulted without waiting for network delay speeds.
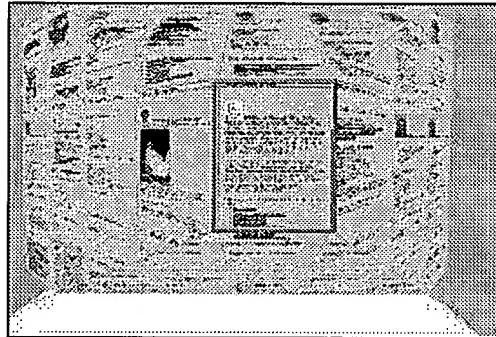
Fig. 4. WebBook viewed with a Document Lens.

Topic Books. A variant is to make sets of books on interesting topics. For example, in our area houses for sale are published on the Web, usually one page per house and usually organized off the home pages of the real-estate brokers. But we have rearranged these ads into houses from a given city. It is then possible rapidly to look back an forth in order to make house comparisons.

Hot List Books. Another variant is hotlist pages. Our system can read any user's Netscape hotlist and automatically fashion it into a set of WebBooks. Although exactly the same URLs are involved, the transformation is dramatic because all the information on all the pages is readily available.

Search Reports. Since WebBooks can be created dynamically, they can be used to display the results of a content based search, either keyword based or relevance feedback (looking for pages similar to a given page). Later results from the search can still be retrieving pages while the user is examining the first pages of the book.

Book Books. A final example comes from observing how some people take multi-page documents and put them on the web. One way to do this is with a series of pages, with next and previous links between pages. When viewing one of these pages with a traditional web browser, there is no indication of how the pages are related. But, such a structure can be easily discovered, and a WebBook constructed from those pages, resulting in a collection of obviously related pages.

Related Work: The Book Metaphor

The book metaphor has been used in both 2D and 3D applications by a number of people for some time. What is new about the WebBook is the integration of an animated 3D book, used to show collections of web pages, with an information workspace manager (the Web Forager) and the application to Web problems.

A book metaphor was chosen after careful examination of a large number of web pages and page collections, and for several reasons: Informally, information on the Web tends to have non-homogeneous character, with closely related weblets situated in more loosely structured environments. An im☐portant subset of these have 'next' and 'previous' links, and thus are thus close operational analogues to books. Furthermore, books as an invention make very efficient use of display space. Starting with a book metaphor, it is easy to escape the serial nature of the physical form by having alternate forms into which the book is transformed, such as the Document Lens in this paper. A book metaphor makes it easy to put actual books on the Internet, something not so at present. We use the book metaphor not primarily because it is familiar, but because of the operational match to a corpus of interest and the efficient display characteriza☐tion. As a bargain, its familiarity allows us to exploit irre☐sistible affordances for low training costs.

Early experiments with giving users access to books, like Brown's Intermedia [10] system in 1985, had limited 3D graphics capabilities. Instead of a simulation of a physical book, Intermedia used 2D layouts of the pages, showing relationships between pages with the layout and with lines drawn between pages. In 1987, Card and Henderson reported the use of a 2D book simulation called "Catalogues" as part of the Rooms system [9], although page turning was not animated. The Xerox TabWorks system [16] was directly inspired by Catalogues. Also in 1987, the BellCore SuperBook Document Browser [13], was designed to transform existing electronic documents into hypertext documents with indexing and a fisheye table of contents, although SuperBook did not use a simulation of a physical book.

Use of a 2D physical book simulation, including page turning, was done in 1987 by Benest [17] for hypertext documents. Similar systems were reported by Miyazawa in 1990 [11] and Ichimura in 1993 [18]. Recently, PenPoint □[14] and General Magic [15] have offered commercial products that use a book metaphor with bookshelves. These are actually 2D animations painted on a background that give the impression of a 3D environment. In 1993, Silicon Graphics introduced the SGI Demo Book [12] as a way of distributing collections of applications, documents, and games. Demo Book is a 3D book that simulates page flipping with animation. The pages hold a collection of icons, 4 rows by 5 columns on each page. The page flipping appears to have flexible pages (broken on the columns). The WebBook page flip currently uses rigid pages, but is displaying text and images instead of rows and columns of icons. Demo Book also has the notion of a bookshelf, although the book always opens to the first page rather than the last page viewed.
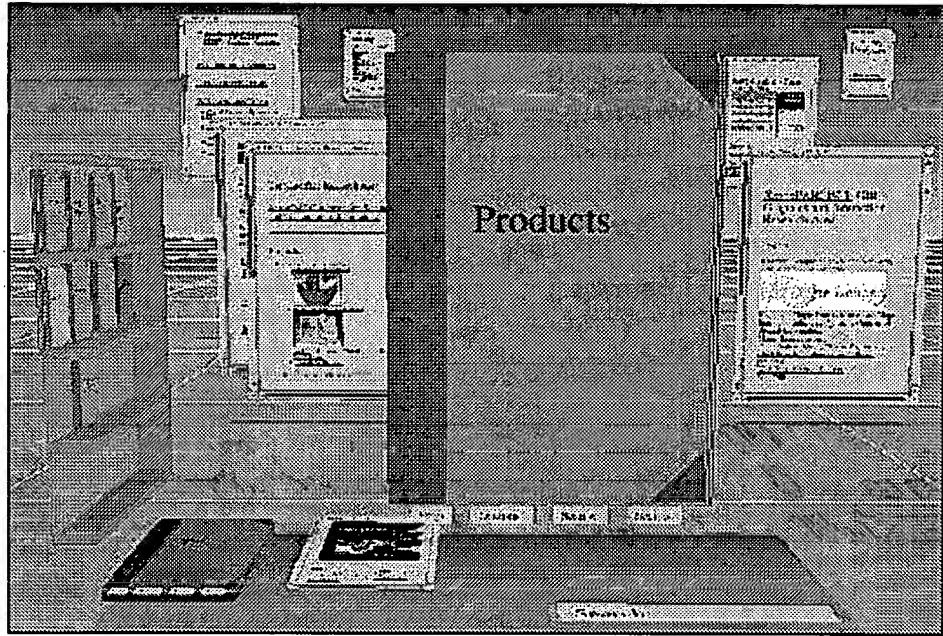

Fig. 5: The Web Forager. (See Color Plate 1)

In 1995, Brown described an experimental web browser, called DeckScape [20], that used a metaphor of decks of cards. Although not a book metaphor, it was an alternative way to solve the problem.

All these systems exploit the human perceptual and cognitive systems by using familiar objects, images and affordances. The page turning of a book conveys information about the relationship of pages, the direction you are moving in the book, the size of the book, and the contents of the book. The WebBook

takes advantage of advances in graphics and processor power to get much closer to a realistic simulation of a book. At the same time, it goes beyond what is possible with a physical book.

THE WEB FORAGER

The WebBook provides a representation of a more aggregate Web entity above the page and allows rapid local interaction with it. The Web Forager allows interaction with multiples of such entities and allows for the necessary tradeoffs among fast access, number of entities, and screen space. The Web Forager is a proposal for a task-tunable information workspace [21] (see Fig. 5).

An individual Web page or a WebBook is presented in a 3D document space (see[22], [23]). Users can click on links in the usual way causing the new linked-to page to fly into the space. The HTML image on the new page develops at the slow Internet speeds (often 15~30 sec). Web pages entered into the space are stored locally and thence forward are available at user interface speeds (around 1~0.1 sec), permitting high interaction rates. These pages can also be grabbed and placed into WebBooks.

Our primary interest in this style of workspace is in exploring the potential for rapid interaction with large numbers of pages. We have previously explored the use of animation, scaling, and 3D-based distortions for building a workspace [21] as a way of handling large numbers of objects. Ark Interface, Inc. [24] produced a pseudo-3D workspace in which functions were associated with parts of a picture of a design studio. Staples [23] did a design mockup study for a 3D user interface. Her goal was to enrich the graphic vocabulary of the workspace by applying perspective, light, and transparency. Ballay [22] with co-workers at the MAYA Design Group implemented a 3D design for an office system. Their design is artistically striking and claimed to be able to handle hundreds of documents. We have sought to break new ground relative to their design on several fronts. First, we tried to increase the speed with which objects can be moved around the space by using gestures. Second, we focused on the Web. Third, the WebBook provides a higher-level object. Fourth, we have experimented with observer movement in the space. And Fifth, we have used a structured model for the generation of the design.

Hierarchical Workspace

The Web Forager workspace (see Fig. 5) is arranged hierarchically (in terms of interaction rates) into three main levels: (1) a Focus Place (the large book or page) showing a page, a book, or an open book at full size for direct interaction between user and content; (2) an Immediate Memory space (the air and the desk), where pages or books can be placed when they are in use, but not the immediate focus (like pages on a desk). A Tertiary Place (the bookcase) where many pages and books can be stored.

The Immediate Storage place has several tiers. Documents are arranged at several distinct z-distances in the 3D space. The user can move these documents back in space (the farther back, the smaller they become and the more documents that fit). Objects in the Immediate Storage place can be moved around in X and Y and moved forward and backward (that is, in Z) using a simple gesture language. A separate Intermediate Storage area is represented by objects on the desk. When the user moves around in the space, the desk, and hence objects on the desk, moves.

The Tertiary Storage area is the bookcase. In normal view, the bookcase is seen at an angle to reduce screen pixels consumed, while at the same time displaying enough information that recently-used books can be recognized. If the user touches one of the books or pages in the bookcase, it will fly up to the focus area (an object occupying that area will automatically fly back to an Immediate Storage position). But if the user wants to examine better the books before making a choice, then touching the bookcase will cause the user, desk and all, to fly to the bookcase (Fig. 6). Touching a book will then cause the user

to fly back to the home position and the book to fly to the focus position.
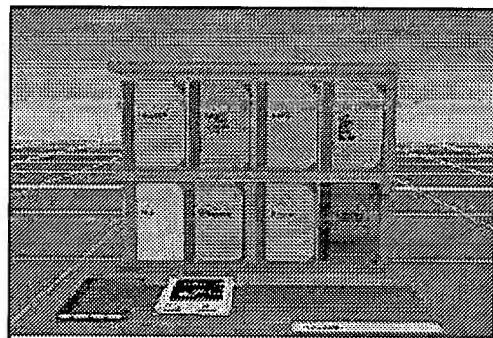
Fig. 6: User has flown to the bookcase where the titles of the books can be easily read.

The purpose of the workspace is to allow a number of objects to be displayed together (mitigating the limitations of the small screen space) in order to support information-intensive activities. The workspace sets up explicitly the capacity for speed/access-time tradeoff to allow tuning for such activities. The design aims to provide very rapid access to a small number of pages, smoothly integrated with the slower access to a large number. The Immediate Storage workspace can hold about 30 documents without occlusion, and over a hundred if occlusions are allowed (not counting book contents). Pages for ongoing activities migrate into the rapid access and manipulation region. In this way, repeated reference due to locality of reference statistics can result in faster interaction.

Cost of Knowledge Characteristic Function

In previous work □[25]□, we have attempted to measure the access properties of a workspace by computing the Cost of Knowledge Characteristic Function, that is, a plot of how many objects can be accessed as a function of the time cost of accessing them. We surmise that a balanced workspace will exhibit an exponential relationship, most conveniently displayed as a straight line in semi-log coordinates. As a tool to use in refining our design, we have computed a preliminary version of this function. The computation assumes that there is a page in the Focus Position (hence the maximal occlusion), that the desk is full, and that one row of pages from each of the discrete Z-distances in the space shows (remember, the design of the space has been carefully set up to permit this).
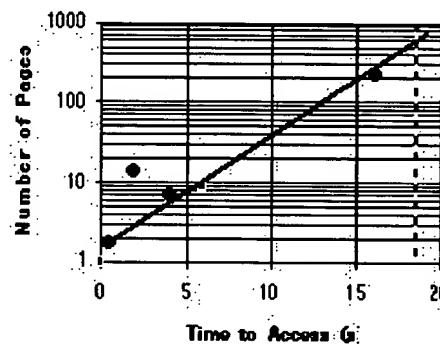

Fig. 7. Preliminary computation of Cost of Knowledge Characteristic Function for Web Forager.

The results are shown in Fig. 7. They are roughly in the expected relation, except that the images on the

desk permit the fast part of the space to receive extra loading. An illustrative comparison with a conventional Web browser is shown as a gray line assuming a constant 18 sec per page retrieval. We hope to use this and related techniques to further refine the space in future research.

SUMMARY

The Web Forager workspace is intended to create patches from the Web where a high density of relevant pages can be combined with rapid access. In addition to multiple pages occurring simultaneously, the space contains groups of pages in the form of WebBooks, which can allow the user to express an elementary form of sensemaking by grouping and ordering. High density patches found around the net, whether created explicitly by searchers or discovered through Web analysis methods can be put into a form where they can be rapidly interacted with. Through the invention of such techniques and analytical methods to help us understand them, it is hoped that the connectivity of the Web, which has been so successful, can be evolved into yet more useful forms.

REFERENCES

1. Pirolli, P. and S. Card, Information Foraging in Information Access Environments, in CHI '95, ACM Conference on Human Factors in Software. 1995, ACM: New York. p. 51Ð58.

2. Stephens, DEW. and J. R. Krebs, Foraging theory. 1986, Princeton: Princeton University Press.

3. Mauldin, M.L. and J.R.R. Leavitt. Web-agent related research at the CMT. in SIGNIDR. 1994. McLean, Virginia:

4. Filo, D. and J. Yang, Yahoo (Yet Another Hierarchical Officious Oracle). 1994, Yahoo! Corp: Stanford, California.

5. D. A. Henderson, J. and S.K. Card, Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interfaces. ACM Transactions on Graphics, 1986. 5(3 (July)): p. 211Ð241.

6. Card, S.K., M. Pavel, and J.E. Farrell, Window-based computer dialogues., in Human-Computer InteractionÑINTERACT '84, B. Shackel, Editor. 1984, Elsevier Science Publishers, B. V. (North-Holland): Amsterdam. p. 51Ð56.

7. Russell, D.M., et al., The cost structure of sensemaking, in ACM/IFIPS InterCHI '3 Conference on Human Factors in Software. 1993, ACM: New York. p. 269Ð276.

8. Robertson, G.G., S.K. Card, and J.D. Mackinlay, Information visualization using 3D interactive animation. Communications of the ACM, 1993. 36(4 (April)): p. 57Ð71.

9. Card, S.K. and D.A. Henderson, Catalogues: A Metaphor for Computer Application Delivery, in Human-Computer Interaction -- INTERACT'87. 1987, Elsevier Science Publishers (North Holland): p. 959Ð963.

10. Yankelovich, N., N. Meyrowitz, and A. van Dam, Reading and Writing the Electronic Book. IEEE Computer, 1985. 18(10 (October)): p. 15Ð30.

11. Miyazawa, M., et al., An Electronic Book: APTBook, in Human-Computer Interaction --

INTERACT'90. 1990, Elsevier Science Publishers (North Holland): Amsterdam. p. 513Ð519.

12. Silicon Graphics, Demo Book. 1993, Silicon Graphics: Mountain View, California.

13. Remde, J.R., L.M. Gomez, and T.K. Landauer, Superbook: An Automatic Tool for Information Exploration, in ACM Hypertext '87 Proceedings. 1987, p. 175Ð188.

14. Carr, R. and D. Shafer, The Power of PenPoint. 1992, New York: Addison-Wesley.

15. Sony Corporation, Magic Link User's Guide, PIC-1000. 1994, Tokyo: Sony Corporation. 203 pp. 16. Moll-Carrillo, H.J., et al., Articulating a Metaphor Through User-Centered Design, in CHI '95, ACM Conference on Human Factors in Software. 1995, ACM Press: New York. p. 566Ð572.

17. Benest, I.D., G. Morgan, and M.D. Smithurst, A Humanized Interface to an Electronic Library, in Human-Computer Interaction -- INTERACT'87. 1987, Elsevier Science Publishers (North Holland): Amsterdam. p. 905Ð910.

18. Ichimura, S. and Y., Another Dimension to Hypermedia Access, in Hypertext '93. 1993, ACM: New York. p. 63Ð72.

19. Robertson, G.G. and J.D. Mackinlay, The Document Lens. UST '93, ACM Conference on User Interface Software and Technology, 1993. : p. 101Ð108.

20. Brown, M.. and R. C. A. Shiner, A New Paradigm for Browsing the Web, in CHI '95, ACM Conference on Human Factors in Software, Conference Companion. 1995, ACM Press: New York. p. 320Ð321.

21. Card, S.K., J.D. Mackinlay, and G.G. Robertson. The Information Visualizer: An information workspace, in CHI '91, ACM Conference on Human Factors in Computing Systems. 1991.

22. Ballay, J.M. Designing Workscape: An interdisciplinary experience. in ACM CHI '94, Human Factors in Computing Systems. 1994. Boston: ACM.

23. Staples, L. Representation in virtual space: Visual convention in the graphical user interface. in INTERCHI '93. 1993. ACM.

24. Ark Interface Workspace User's Guide. 1990, Seattle, Washington: Ark Interface, Inc. 54.

25. Card, S.K., P. Pirolli, and J.D. Mackinlay, The cost of knowledge characteristic function: Display evaluation for direct-walk dynamic information visualizations, in CHI '94, ACM Conference on Human Factors in Software. 1994, ACM: New York. p. 238Ð244.

Perlin Noise

Many people have used random number generators in their programs to create unpredictability, make the motion and behavior of objects appear more natural, or generate textures. Random number generators certainly have their uses, but at times their output can be too harsh to appear natural. This article will present a function which has a very wide range of uses, more than I can think of, but basically anywhere where you need something to look natural in origin. What's more it's output can easily be tailored to suit your needs.

If you look at many things in nature, you will notice that they are fractal. They have various levels of detail. A common example is the outline of a mountain range. It contains large variations in height (the mountains), medium variations (hills), small variations (boulders), tiny variations (stones) . . . you could go on. Look at almost anything: the distribution of patchy grass on a field, waves in the sea, the movements of an ant, the movement of branches of a tree, patterns in marble, winds. All these phenomena exhibit the same pattern of large and small variations. The Perlin Noise function recreates this by simply adding up noisy functions at a range of different scales.

To create a Perlin noise function, you will need two things, a Noise Function, and an Interpolation Function.

Introduction To Noise Functions

A noise function is essentially a seeded random number generator. It takes an integer as a parameter, and returns a random number based on that parameter. If you pass it the same parameter twice, it produces the same number twice. It is very important that it behaves in this way, otherwise the Perlin function will simply produce nonsense.
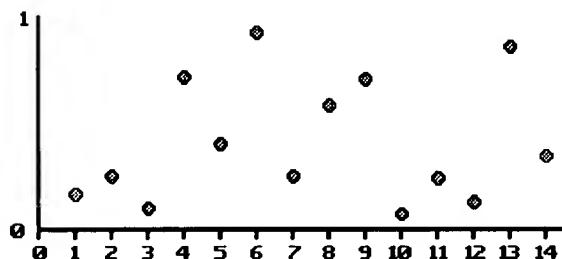

Here is a graph showing an example noise function. A random value between 0 and 1 is assigned to every point on the X axis.
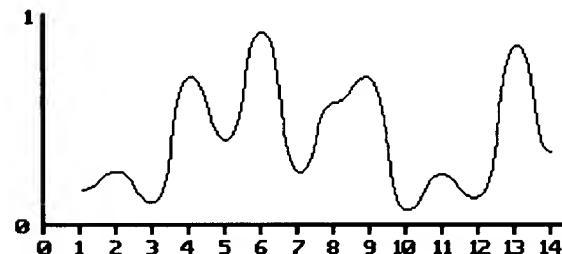
By smoothly interpolating between the values, we can define a continuous function that takes a non-integer as a parameter. I will discuss various ways of interpolating the values later in this article.

Definitions

Before I go any further, let me define what I mean by **amplitude** and **frequency**. If you have studied physics, you may well have come across the concept of amplitude and frequency applied to a sin wave.
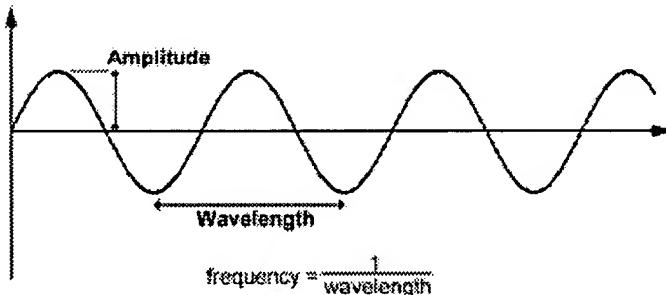
Sin Wave

The wavelength of a sin wave is the distance from one peak to another. The amplitude is the height of the wave. The frequency is defined to be $1/_{\text{wavelength}}$.
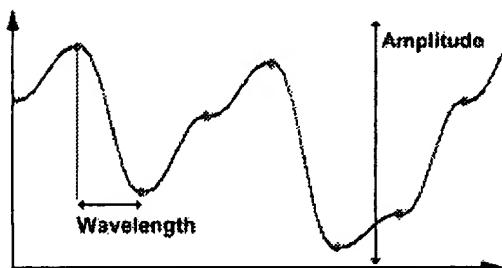
Noise Wave

In the graph of this example noise function, the red spots indicate the random values defined along the dimension of the function. In this case, the amplitude is the difference between the minimum and maximum values the function could have. The wavelength is the distance from one red spot to the next. Again frequency is defined to be $1/_{\text{wavelength}}$.

Creating the Perlin Noise Function

Now, if you take lots of such smooth functions, with various frequencies and amplitudes, you can add them all together to create a nice noisy function. This is the Perlin Noise Function.

### Take the following Noise Functions

Amplitude : 128
frequency : 4

Amplitude : 64
frequency : 8

Amplitude : 32
frequency : 16

Amplitude : 16
frequency : 32
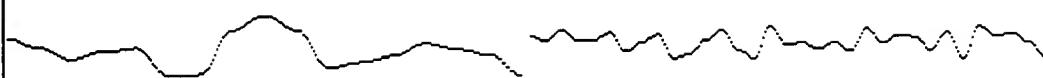
Amplitude : 8
frequency : 64

Amplitude : 4
frequency : 128
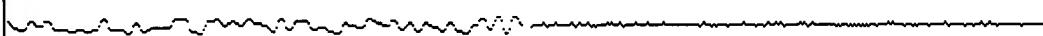
Sum of Noise Functions = (Perlin Noise)

Add them together, and this is what you get.

You can see that this function has large, medium and small variations. You may even imagine that it looks a little like a mountain range. In fact many computer generated landscapes are made using this method. Of course they use 2D noise, which I shall get onto in a moment.

You can, of course, do the same in 2 dimensions.

Some noise functions are created in 2D
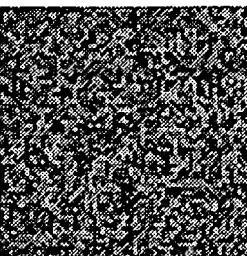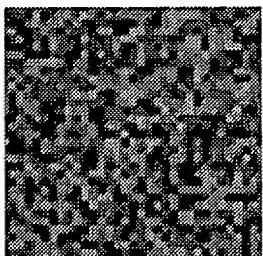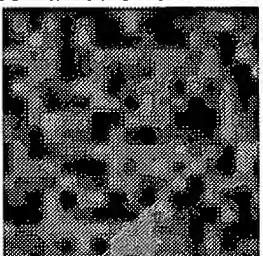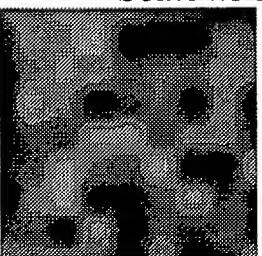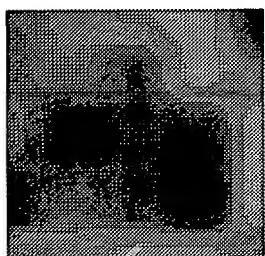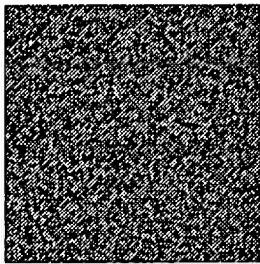
Adding all these functions together produces a noisy pattern.
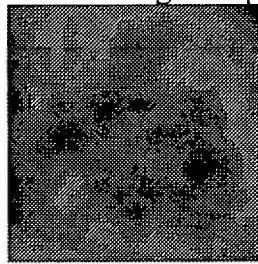
Persistence

When you're adding together these noise functions, you may wonder exactly what amplitude and frequency to use for each one. The one dimensional example above used twice the frequency and half the amplitude for each successive noise function added. This is quite common. So common in fact, that many people don't even consider using anything else. However, you can create Perlin Noise functions with different characteristics by using other frequencies and amplitudes at each step. For example, to create smooth rolling hills, you could use Perlin noise function with large amplitudes for the low frequencies , and very small amplitudes for the higher frequencies. Or you could make a flat, but very rocky plane choosing low amplitudes for low frequencies.

To make it simpler, and to avoid repeating the words Amplitude and Frequency all the time, a single number is used to specify the amplitude of each frequency. This value is known as **Persistence**. There is some ambiguity as to it's exact meaning. The term was originally coined by Mandelbrot, one of the people behind the discovery of fractals. He defined noise with a lot of high frequency as having a low persistence. My friend Matt also came up with the concept of persistence, but defined it the other way round. To be honest, I prefer Matt's definition. Sorry Mandelbrot. So our definition of persistence is this:

```
frequency = 2ⁱ
amplitude = persistenceⁱ
```

$$frequency = 2^i$$
$$amplitude = persistence^i$$

Where i is the ith noise function being added. To illustrate the effect of persistence on the output of the Perlin Noise, take a look at the diagrams below. They show the component noise functions that are added, the effect of the persistence value, and the resultant Perlin noise function.

Frequency	1	2	4	8	16	32	
Persistence		+	+	+	+	+	=

= 1/4

| Amplitude: | 1 | $\frac{1}{4}$ | $\frac{1}{16}$ | $\frac{1}{64}$ | $\frac{1}{256}$ | $\frac{1}{1024}$ | result |

Persistence = 1/2

| Amplitude: | 1 | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | result |

Persistence = 1 / root2

| Amplitude: | 1 | $\frac{1}{1.414}$ | $\frac{1}{2}$ | $\frac{1}{2.828}$ | $\frac{1}{4}$ | $\frac{1}{5.656}$ | result |

Persistence = 1

| Amplitude: | 1 | 1 | 1 | 1 | 1 | 1 | result |
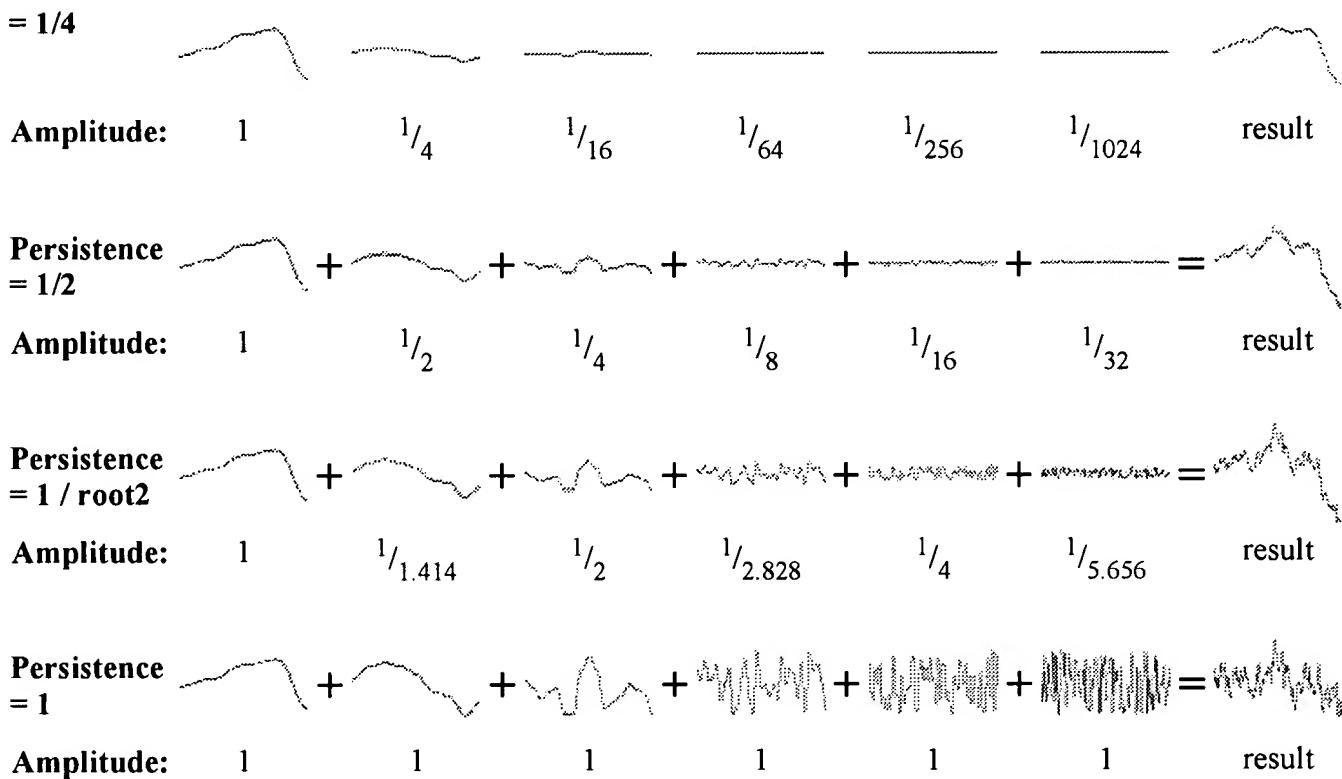
Octaves

Each successive noise function you add is known as an **octave**. The reason for this is that each noise function is twice the frequency of the previous one. In music, octaves also have this property.

Exactly how many octaves you add together is entirely up to you. You may add as many or as few as you want. However, let me give you some suggestions. If you are using the perlin noise function to render an image to the screen, there will come a point when an octave has too high a frequency to be displayable. There simply may not be enough pixels on the screen to reproduce all the little details of a very high frequency noise function. Some implementations of Perlin Noise automatically add up as many noise functions they can until the limits of the screen (or other medium) are reached.

It is also wise to stop adding noise functions when their amplitude becomes too small to reproduce. Exactly when that happens depends on the level of persistence, the overall amplitude of the Perlin function and the bit resolution of your screen (or whatever).

Making your noise functions

What do we look for in a noise function? Well, it's essentially a random number generator. However, unlike other random number generators you may have come across in your programs which give you a different random number every time you call them, these noise functions supply a random number calculated from one or more parameters. I.e. every time you pass the same number to the noise function, it will respond with the same number. But pass it a different number, and it will return a different number.

Well, I don't know a lot about random number generators, so I went looking for some, and here's one I found. It seems to be pretty good. It returns floating point numbers between -1.0 and 1.0.

```
function IntNoise(32-bit integer: x)

    x = (x<<13) ^ x;
    return ( 1.0 - ( (x * (x * x * 15731 + 789221) + 1376312589) & 7fffffff) / 10737

end IntNoise function
```

Now, you'll want several different random number generators, so I suggest making several copies of the above code, but use slightly different numbers. Those big scarey looking numbers are all prime numbers, so you could just use some other prime numbers of a similar size. So, to make it easy for you to find random numbers, I have written a little program to list prime numbers for you. You can give it a start number and an end number, and it will find all the primes between the two. Source code is also included, so you can easily include it into your own programs to produce a random prime number. **Primes.zip**

Interpolation

Having created your noise function, you will need to smooth out the values it returns. Again, you can choose any method you like, but some look better than others. A standard interpolation function takes three inputs, **a** and **b**, the values to be interpolated between, and **x** which takes a value between 0 and 1. The Interpolation function returns a value between **a** and **b** based on the value **x**. When **x** equals 0, it returns **a**, and when **x** is 1, it returns **b**. When **x** is between 0 and 1, it returns some value between **a** and **b**.

Linear Interpolation:

Looks awful, like those cheap 'plasmas' that everyone uses to generate landscapes. It's a simple algorithm though, and I suppose would be excusable if you were trying to do perlin noise in realtime.

```
function Linear_Interpolate(a, b, x)
       return  a*(1-x) + b*x
end of function
```
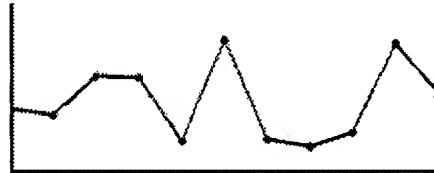
Cosine Interpolation:

This method gives a much smother curve than Linear Interpolation. It's clearly better and worth the effort if you can afford the very slight loss in speed.

```
function Cosine_Interpolate(a, b, x)
       ft = x * 3.1415927
       f = (1 - cos(ft)) * .5

       return  a*(1-f) + b*f
end of function
```

Cubic Interpolation:

This method gives very smooth results indeed, but you pay for it in speed. To be quite honest, I'm not sure if it would give noticeably better results than Cosine Interpolation, but here it is anyway if you want it. It's a little more complicated, so pay attention. Whereas before, the interpolation functions took three inputs, the cubic interpolation takes five. Instead of just **a** and **b**, you

now need **v0, v1, v2** and **v3**, along with **x** as before. These are:

v0 = the point before **a**
v1 = the point **a**
v2 = the point **b**
v3 = the point after **b**
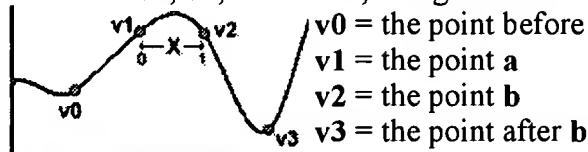
```
function Cubic_Interpolate(v0, v1, v2, v3,x)
        P = (v3 - v2) - (v0 - v1)
        Q = (v0 - v1) - P
        R = v2 - v0
        S = v1

        return Px³ + Qx² + Rx + S
end of function
```
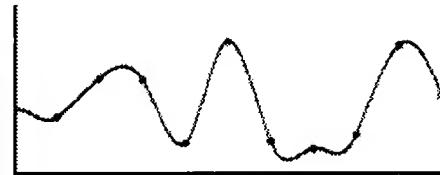
Smoothed Noise

Aside from Interplolation, you can also smooth the output of the noise function to make it less random looking, and also less square in the 2D and 3D versions. Smoothing is done much as you would expect, and anyone who has written an image smoothing filter, or fire algorithm should already be familiar with the process.

Rather than simply taking the value of the noise function at a single coordinate, you can take the average of that value, and it's neighbouring values. If this is unclear, take a look at the pseudo code below.

On the right, you can see a little diagram illustrating the difference between smoothed noise, and the same noise function without smoothing. You can see that the smooth noise is flatter, never reaching the extremes of unsmoothed noise, and the frequency appears to be roughly half. There is little point smoothing 1 dimensional noise, since these are really the only effects. Smoothing becomes more useful in 2 or three dimensions, where the effect is to reduce the squareness of the noise. Unfortunately it also reduces the contrast a little. The smoother you make it, obviously, the flatterthe noise will be.
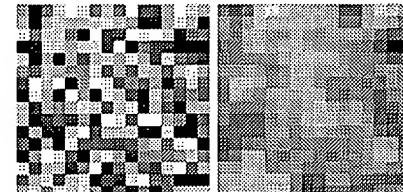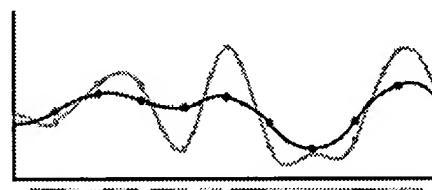
1-dimensional Smooth Noise

```
function Noise(x)
      .
      .
end function

function SmoothNoise_1D(x)

   return Noise(x)/2  +  Noise(x-1)/4  +  Noise(x+1)/4

end function
```

2-dimensional Smooth Noise

```
function Noise(x, y)
```

```
end function

function SmoothNoise_2D(x>, y)

  corners = ( Noise(x-1, y-1)+Noise(x+1, y-1)+Noise(x-1, y+1)+Noise(x+1, y+1) ) /
  sides   = ( Noise(x-1, y)  +Noise(x+1, y)  +Noise(x, y-1)  +Noise(x, y+1) ) /  8
  center  =  Noise(x, y) / 4

  return corners + sides + center


end function
```

Putting it all together

Now that you know all that, it's time to put together all you've learned and create a Perlin Noise function. Remember that it's just several Interpolated Noise functions added together. So Perlin Noise it just a function. You pass it one or more parameters, and it responds with a number. So, here's a simple 1 dimensional Perlin function.

The main part of the Perlin function is the loop. Each iteration of the loop adds another octave of twice the frequency. Each iteration calls a *different* noise function, denoted by **Noise**$_i$. Now, you needn't actually write lots of noise functions, one for each octave, as the pseudo code seems to suggest. Since all the noise functions are essentially the same, except for the values of those three big prime numbers, you can keep the same code, but simply use a different set of prime numbers for each.

1-dimensional Perlin Noise Pseudo code

```
function Noise1(integer x)
  x = (x<<13) ^ x;
  return ( 1.0 - ( (x * (x * x * 15731 + 789221) + 1376312589) & 7fffffff) / 10737
end function


function SmoothedNoise_1(float x)
  return Noise(x)/2  +  Noise(x-1)/4  +  Noise(x+1)/4
end function


function InterpolatedNoise_1(float x)

    integer_X    = int(x)
    fractional_X = x - integer_X

    v1 = SmoothedNoise1(integer_X)
    v2 = SmoothedNoise1(integer_X + 1)

    return Interpolate(v1 , v2 , fractional_X)

end function


function PerlinNoise_1D(float x)
```

```
        total = 0
        p = persistence
        n = Number_Of_Octaves - 1

        loop i from 0 to n

            frequency = 2ⁱ
            amplitude = pⁱ

            total = total + InterpolatedNoiseᵢ(x * frequency) * amplitude

        end of i loop

        return total

end function
```

Now it's easy to apply the same code to create a 2 or more dimensional Perlin Noise function:

2-dimensional Perlin Noise Pseudocode

```
function Noise1(integer x, integer y)
  n = x + y * 57
  n = (n<<13) ^ n;
  return ( 1.0 - ( (n * (n * n * 15731 + 789221) + 1376312589) & 7fffffff) / 10737
end function

function SmoothNoise_1(float x, float y)
  corners = ( Noise(x-1, y-1)+Noise(x+1, y-1)+Noise(x-1, y+1)+Noise(x+1, y+1) ) /
  sides   = ( Noise(x-1, y)  +Noise(x+1, y)  +Noise(x, y-1)  +Noise(x, y+1) ) /  8
  center  =   Noise(x, y) / 4
  return corners + sides + center
end function

function InterpolatedNoise_1(float x, float y)

    integer_X    = int(x)
    fractional_X = x - integer_X

    integer_Y    = int(y)
    fractional_Y = y - integer_Y

    v1 = SmoothedNoise1(integer_X,     integer_Y)
    v2 = SmoothedNoise1(integer_X + 1, integer_Y)
    v3 = SmoothedNoise1(integer_X,     integer_Y + 1)
    v4 = SmoothedNoise1(integer_X + 1, integer_Y + 1)

    i1 = Interpolate(v1 , v2 , fractional_X)
    i2 = Interpolate(v3 , v4 , fractional_X)

    return Interpolate(i1 , i2 , fractional_Y)

end function


function PerlinNoise_2D(float x, float y)
```

```
    total = 0
    p = persistence
    n = Number_Of_Octaves - 1

    loop i from 0 to n

        frequency = 2^i
        amplitude = p^i

        total = total + InterpolatedNoise_i(x * frequency, y * frequency) * amplitu

    end of i loop

    return total

end function
```

Applications of Perlin Noise

Now that you have this fantastic function, what can you do with it? Well, as the cliche goes, you're limited only by your imagination. Perlin Noise has so many applications that I can't think of them all, but I'll have a go.

1 dimensional

Controlling virtual beings: Living objects rarely stay still for very long (except students). Use perlin noise to constantly adjust the joint positions of a virtual human player, in a game for example, to make it look like it's more alive.

Drawing sketched lines: Computer drawn lines are always totally straight, which can make them look unnatural and unfriendly. You can use Perlin Noise to introduce a wobblyness to a line drawing algorithm to make it appear as if it's been drawn by hand. You can also draw wobbly circles and boxes. Some research has been done on making a Sketchy User Interface.
See: <u>Creating Informal Looking Interfaces</u>.
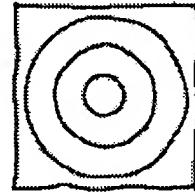
2 dimensional

Landscapes: These are a perfect application for 2D Perlin Noise. Unlike the subdivision method, you do not have to store the landscape anywhere in memory, the height of any point on the landscape can be calculated easily. What's more, the land stretches indefinitely (almost), and can be calculated to minute detail, so it's perfect of variable level of detail rendering. The properties of the landscape can be defined easily too.

Clouds: Again, cloud rendering is well suited to Perlin Noise.

Generating Textures: All sorts of textures can be generated using Perlin Noise. See the table below for some examples. The textures generated can go on for ages before repeating (if ever), which makes them much more pleasant to look at than a repeating tiled texture map.

3 dimensional

3D Clouds: You can, of course, produce volumetric clouds. You'll probably have to use some sort of ray tracing to visualise them.

Animated Clouds: You can produce animated 2 dimensional clouds with 3D Perlin Noise, if you consider one dimension to be time.

Solid Textures: Some rendering / raytracing programs, like POVray, apply texture to objects by literally carving them from a 3-dimensional texture. This was, the textures do not suffer from the warping usually associated with mapping 2D textures onto (non-flat) 3D objects.

4 dimensional

Animated 3D Textures and Clouds: Moving into higher dimensions, you can easily produce animated clouds and solid textures. Just consider the extra dimension to be time.

Copyright Matt Fairclough 1998

The land, clouds and water in this picture were all mathematically generated with Perlin Noise, and rendered with Terragen.
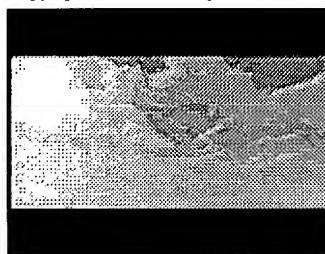
The clouds in this demo are animated with 3D perlin Noise. The algorithm had to be modified slightly to be able to produce Perlin Noise in real time. See the Clouds Article for more info on how this was done.

Generating Textures with Perlin Noise

Perlin is fantastic for generating textures. You can produce textures that are (for all practical purposes) infinitely large, but take up almost no memory. You can create marble, wood, swirly patterns, probably anything if you try hard. You can also define a 3D texture. You can think of this as a solid

block of material, from which you can 'carve' an object. This allows you to produce textures which can be applied to any shaped object without distortion. It can take a lot of imagination, thought and experimentation to get a texture to look really good, but the results can be very impressive indeed.

Play around as much as you like. Use several Perlin functions to create a texture, try different persistences and different frequencies in different dimensions. You can use one Perlin function to affect the properties of another. Apply functions to their output. Do whatever you want, there's almost certainly a way to produce almost any texture you can dream up.

The following textures were made with 3D Perlin Noise

	Standard 3 dimensional perlin noise. 4 octaves, persistence 0.25 and 0.5
	Low persistence. You can create harder edges to the perlin noise by applying a function to the output.
	To create more interesting and complicated textures, you should try mixing several Perlin functions. This texture was created in two parts. Firstly a Perlin function with low persistence was used to define the shape of the blobs. The value of this function was used to select from two other functions, one of which defined the stripes, the other defined the blotchy pattern. A high value chose more of the former, a low value more of the latter. The stripes were defined by multiplying the first Perlin Function by some number (about 20) then taking the cosine.
	A marbly texture can be made by using a Perlin function as an offset to a cosine function. `texture = cosine( x + perlin(x,y,z) )`
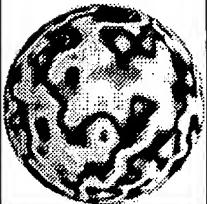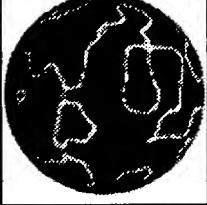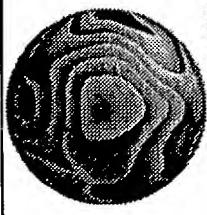
	Very nice wood textures can be defined. The grain is defined with a low persistence function like this:
	
	`g = perlin(x,y,z) * 20`
	`grain = g - int(g)`
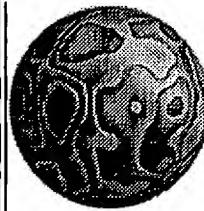
The very fine bumps you can see on the wood are high frequency noise that has been stretched in one dimension.

```
bumps = perlin(x*50, y*50, z*20)
if bumps < .5 then bumps = 0  else bumps = 1t
```

References

Procedural textures: **http://developer.intel.com/drg/mmx/appnotes/proctex.htm**

Intel Developer Site article about using the new MMX technology to render Perlin Noise in real time.

Ken Perlin's Homepage: **http://mrl.nyu.edu/perlin/**

I assume the person responsable for Perlin Noise. He has an interesting page with lots of useful links to texturing and modeling stuff.

Texturing And Modeling A Procedural Approach:
http://www.cs.umbc.edu/~ebert/book/book.html

Ken Perlin's book which goes in depth on using Perlin Noise, among other algorithms to generate textures and model various natural phenomena.

Procedural Texture Page: **http://www.threedgraphics.com/pixelloom/tex_synth.html**

This page is an attempt to collect any and all information and WWW links related to Procedural Texture Synthesis.

Return to the Good Looking Textured Light Sourced Bouncy Fun Smart and Stretchy Page.

HOTBOT

| perlin noise function | Homepage | Advanced Search |

Search using: **HotBot**   **Google**   **Ask Jeeves**

CUSTOM WEB FILTERS   Tools | HotBot Skins

Date: **Before February 14 2000** [Edit this Search]

SPONSORED LINKS (filters not applied)

- **Noise problems solved**
  Learn what the pro's use. Expert advice.
  www.quietsolution.com

WEB RESULTS (Showing Results 1 - 10 of 119)

1. Perlin Noise
... pattern of large and small variations. The **Perlin Noise function** recreates this by simply ad noisy functions ...
freespace.virgin.net/hugo.elias/models/m_perlin.htm - December 7, 1998 - 35 KB

2. Perlin Noise in Animation
... **Perlin Noise** in Animation. My goal for this project was to develop a tool that allows an anim came from the application of a **noise function** to his pelvis joint. ...
www.cs.wisc.edu/graphics/Courses/cs-838-1999/Students/fruit/final_writeup.h - May 14, 1999 -

3. Terrain -- Noise Synthesis
... have ported to Java. The **noise function** in this program differs significantly from **Perlin no** worth studying. ...
www.geocities.com/Area51/6902/t_nsyn.html - June 16, 1998 - 6 KB

4. Project 6 - Solid State Texturing
Project 6 - Solid State Texturing - Matthew Kaplan Here is a solid texture stripe images at 100 s Here are a bunch of images I produced using the **Perlin noise function**. All of the textured ...
www.cs.utah.edu/~kaplan/imgsyn/project6.html - September 25, 1999 - 3 KB

5. 1 Introduction
Implicit Representations of Rough Surfaces John C. Hart School of EECS Washington State Univ Pullman, WA 99164-2752 Implicit surface techniques provide useful tools for modeling and ... .. limited random signal [**Perlin**, 1985]. Although random, the **noise function** is smooth. ...
graphics.cs.uiuc.edu/~jch/papers/rough.pdf - January 22, 2000 - 184 KB

6. han:s1
A Cellular Texture Basis **Function** New Space Partitioning Basis **Function** for Textured Surface Compliments Perlin's **Noise Function** Computed On the Fly Produces a Range of Effects
graphics.lcs.mit.edu/~mcm/6.838j/worley/s1.html - March 21, 1997 - 1 KB

7. Radiance Digest v2n7
... is that noise3(x y z) is the **Perlin noise function** and fnoise3(x y z) is a fractal **noise funct** otherwise similar ...
radsite.lbl.gov/radiance/digests_html/v2n7.html - February 18, 1998 - 41 KB

8. untitled
... (x#, y#, z#) 'procedural marble DECLARE **FUNCTION noise#** (x#, y#, z#) '3d **Perlin nois**

'x# y# z# are a point in ...
www.allbasiccode.com/files/abc9811/realray.bas - January 10, 1999 - 5 KB

9. Shaders
Using and Writing Shaders All color, displacement, contour, and other computation in mental ra shaders.
www.qarl.com/menu/class/cs323_fl98/mrmanual_2.0/shaders.html - November 27, 1999 - 525 KB

10. Microsoft Word - pxflshading.
... more complex operations like a **Perlin noise function.** The hard- ...
www.cs.unc.edu/~pxfl/papers/pxflshading.pdf - June 11, 1998 - 170 KB

« Previous | Next »

Search for "**perlin noise function** " using: Google,   Ask Jeeves

Portions powered by ⬡ Inktomi

HOTBOT

| Interface Ecology | Homepage | Advanced Search |

Search using: **HotBot**   **Google**   **Ask Jeeves**

CUSTOM WEB FILTERS   Tools | HotBot Skins

Date: **Before February 14 2000** [Edit this Search]

SPONSORED LINKS (filters not applied)

- **Ecology** Articles
  The New York Times is a valuable resource for students and faculty.
  www.nytimes.com/college

WEB RESULTS (Showing Results 1 - 10 of 4,446)

1. CHI 97: CollageMachine: Temporality and Indeterminacy in Media...
... science techniques into this interactive environment arises through application of the theory
Ecology. ...
www.acm.org/sigchi/chi97/proceedings/short-talk/ak.htm - August 11, 1997 - 12 KB

2. CHI 97: CollageMachine: Temporality and Indeterminacy in Media...
CHI 97 Electronic Publications: Late-Breaking/Interactive Posters CollageMachine: Temporality a
Indeterminacy in Media Browsing via **Interface Ecology** (poster) © 1997 Copyright on this ma
www.acm.org/sigchi/chi97/proceedings/poster/ak.htm - August 11, 1997 - 2 KB

3. Conservation **Ecology**: Embracing uncertainty: The **interface** of...
Judith L. Anderson. 1998. Embracing uncertainty: The **interface** of Bayesian statistics and cogn
psychology. Conservation **Ecology** [online]2(1): 2.
www.consecol.org/Journal/vol2/iss1/art2 - January 7, 2000 - 109 KB

4. untitled
COLLEGE OF LIBERAL ARTS AND SCIENCES African and African-American Studies Courses A 21
1.1.2211 AF S 104 The Peoples of Africa (3) WS SC NW.
www.ukans.edu/cwis/courses/Descriptions.html - October 14, 1993 - 524 KB

5. Anba Tonnel - Environment & **Ecology** in Haiti; The Online Discussion...
Online Discussion Forum on the **ecology** of Haiti in Haiti Global Village. Haiti Global Village, the
Interface to the Internet, is a beautifully designed online community devoted to Haiti ......
www.haitiglobalvillage.com/sd-at-ecology - May 10, 1999 - 4 KB

6. WORKSHOP ON COMPUTATIONAL **ECOLOGY**: Agenda
Participant Write-ups & Biographies We ask all participants provide a one-page write-up concern
interests as related to the purpose of the workshop. A short bio. is also requested but not ... ...
pest management, community **ecology** and island biogeographic theory, ... design theory and
ecology, and has worked extensively ... we teach an **ecology** class we aren't ...
www.sdsc.edu/compeco_workshop/writups.html - October 11, 1995 - 69 KB

7. ACADEMIC PLAN SECTION OF EVOLUTION AND **ECOLOGY** DIVISION OF BIOLOG
SECTION OF EVOLUTION AND **ECOLOGY** DIVISION OF BIOLOGICAL SCIENCES Research and te
activities in the Section of Evolution and **Ecology**, as the name implies, focus on the two basic
disciplines ......
www.dbs.ucdavis.edu/about/plan/eve.pdf - June 30, 1999 - 41 KB

8. Who's Zooming Whom? Attunement to Animation in the **Interface**
Who's Zooming Whom? Attunement to Animation in the **Interface** Computer Science Departme
Cognitive Science Program Lindley Hall 215 Bloomington, Indiana 47405 School of Library and
Information ... ... Apple Macintosh graphical user **interface** aids in a user's ... users with the M
interface and the zooming effect. ...
www.cs.indiana.edu/hyplan/mchui/zoom.html - September 17, 1996 - 29 KB

9. **ECOLOGY** GRADUATE STUDENTS
... at this time lie on the **interface** between physiology and ecosystem **ecology**. I am studying
physiological responses ...
lsvl.la.asu.edu/ecology/grads.html - November 10, 1999 - 11 KB

10. Animal **Ecology** Faculty in EEB
Below are the Animal **Ecology** Department faculty members participating in the **Ecology** and E
interdepartmental major. Clicking on the highlighted links takes you directly to an individual ...
Bowen's research interests lie at the **interface** of behavior, **ecology**, evolution, and genetics. .
www.public.iastate.edu/~eeb/AECOL.html - October 7, 1999 - 29 KB

« Previous | Next »

Search for "**Interface Ecology**" using: Google, Ask Jeeves

Portions powered by ⬡ I n k t o m i

combin
Formation

an instance of interface ecology

about

combinFormation provides a generative information space for browsing, collecting, and organizing information samples from the net. By information space, we mean a place where you can arrange the samples in space, visually. By generative, we mean that there is an agent that works with you on developing the space. The agent tries to understand your interests, and work cooperatively with you on the space. To do this it needs to understand what you are interested in. We provide tools for expressing design and interest intentions, at the same time.

generative agent

The generative agent doesn't wait for you to click through links. It composes the space procedurally, based on its model of your interests. It includes a web crawler, that will run through links automatically. The agent will carve the web pages it finds up into sets of information samples. It will periodically place a sample into the visual space for you. When the space gets filled, it will also remove elements. You can turn the agent on and off, and control its speed, using the tape recorder controls.

information samples

The information samples may be images and small chunks of text. Information samples that come from the net include references back to their

original web pages. They also may include hyperlinks to other pages. The information space enables you to navigate from an element to its container, and any hyperlink(s). In this way it works as a graphical collection of bookmarks.

recombinant information

element rollover

You can also see each element distinctly by placing your mouse over it. At the same time, descriptive information will be displayed.

what-its-for

You can express yourself with the program to create information spaces on a topic. These are visual collections of information samples. The spaces may help you get a fresh perspective on materials with which your already familiar. You can also share these information spaces with others by publishing them on the web. The information space will act as a special sort of active web page. The people who browse your information spaces will be able to modify them with the same tools you have. They can navigate back to the original and hyperlinked documents. They can use the generative capability to bring in more stuff. They can publish their own derivative versions.

The program can also be used by presenters of collections of internet content, especially when it is rich in images. Presenters, such as digital libraries, and catalogs, may wish to use the generative information space to provide active and fresh views of their collections.

references

Kerne, A., Sundaram, V., Wang, J., Khandelwal, M., Mistrot, J.M. **Human + Agent: Creating Recombinant Information**, *Proc ACM Multimedia 2003*, in press.

Kerne, A., Sundaram, V. **A Recombinant Information Space**, *Proc Computational Semiotics in Games and New Media (CoSIGN) 2003, 48-57,*

Kerne, A., Khandelwal, M., Sundaram, V. **Publishing Evolving Metadocuments on the Web**. *Proc ACM Hypertext 2003*, 104-105.

Kerne, A. **Concept-Context-Design: A Creative Model for the Development of Interactivity**. *Proc ACM Creativity and Cognition 2002*, 192-199.

CollageMachine: **Interest-Driven Browsing Through Streaming Collage**. *Proc Cast01: Living in Mixed Realities*. Bonn, Germany, 241-244.

CollageMachine: **A Model of "Interface Ecology"**. NYU Ph.D. Dissertation. April 2001.

CollageMachine: An Interactive Agent of Web Recombination.
Leonardo MIT Press, 3:5, Nov 2000, 347-350.

CollageMachine: Temporality and Indeterminacy in Media Browsing via Interface Ecology. *Proc CHI 1997: Extended,* 297-298.